17. Advanced Web Attack Techniques

🔥 Impact and Risk of Web Application Vulnerabilities

🔻 Impacts:

- **Data Theft**: Sensitive info like session tokens, user credentials.
- **Session Hijacking**: Attackers impersonate users/admins.
- **Website Defacement**: Alter content for propaganda or damage.
- **Malware Delivery**: Injecting malicious scripts into trusted sites.
- **Reputation Damage**: Loss of customer trust, media backlash.
- **Legal & Compliance Issues**: Violations of GDPR, HIPAA, etc.

⚠️ Risks by Vulnerability Type:

Vulnerability	Risk Description
XSS	Steals cookies/session, redirects, defaces pages
SQL Injection	Bypasses logins, dumps DBs, modifies data
CSRF	Forces users to perform actions like money transfers
IDOR	Exposes or manipulates unauthorized data

🧠 Identify and Explain Types of XSS (Cross-Site Scripting)

1. Stored XSS:

- Malicious input is saved (e.g., comment section), and served to users.
- **Impact**: Every visitor gets the malicious payload.

2. Reflected XSS:

- Payload is reflected immediately in the HTTP response (e.g., URL parameter).
- **Impact**: Requires a victim to click a crafted link.

3. DOM-Based XSS:

- Vulnerability lies in client-side JavaScript manipulating DOM unsafely.
- **Impact**: Script executes directly in the browser environment.

🏴‍☠️ How XSS Injections Work

Basic Example (Reflected XSS):

```
https://example.com/search?q=<script>alert('XSS')</script>
```

- The site reflects `q` in HTML **without sanitizing it**.
- Browser executes the script → **alert pops** → in real attack: steals cookies, logs keystrokes, etc.

🛡️ Implement Prevention Techniques

✅ Best Practices:

Method	How it Helps
Input Validation	Reject or sanitize bad inputs early.
Output Encoding	Encode special characters (`<`, `>`, `'`, `"`) in HTML/JS.
Content Security Policy (CSP)	Restricts what scripts can run.
HTTPOnly Cookies	Prevents JavaScript from accessing cookies.
Escaping User Inputs	Encode data depending on context (HTML, JS, URL, etc).

🧪 Tools to Use:

- `OWASP ZAP`, `Burp Suite`, browser dev tools for testing
- Libraries: DOMPurify, Google Caja for JS sanitization

👨‍💻 Recognize the Importance of Secure Coding Practices

- **Failing to code securely leads to predictable and repeatable exploits.**
- Secure code = **cheaper to maintain**, **easier to audit**, and **harder to exploit**.
- Think like a threat actor when writing or reviewing code.

Examples:

- Use **prepared statements** (not string concat) for SQL queries.
- Validate input on **both client and server sides**.
- Escape user data **at the point of use**.

🔄 Integrate Security into the Development Lifecycle (DevSecOps)

🔁 Steps to Follow:

1. **Threat Modeling** during planning.
2. **Static Code Analysis** (e.g., SonarQube, Checkmarx) in CI/CD.
3. **Dynamic App Testing (DAST)** during staging.
4. **Security Code Reviews** as part of pull requests.

5. **Regular Pentesting** and bug bounty participation.
6. **Security Awareness Training** for developers.

📚 Summary

Concept	Description
XSS Types	Stored, Reflected, DOM-based
How XSS Works	Injects JS into trusted site context
Impact of Web Vulns	Theft, defacement, hijacking, legal issues
Mitigation	Input validation, output encoding, CSP
Secure Coding	Validations, escaping, using safe APIs
DevSecOps Integration	Build security into every stage of SDLC
